ELSEVIER

Discrete Applied Mathematics 60 (1995) 159–179

DISCRETE APPLIED MATHEMATICS

A new algorithm for the propositional satisfiability problem

G. Gallo*, D. Pretolani

Dipartimento di Informatica, University of Pisa, Corso Italia 40, 56125 Pisa, Italy

Received 28 October 1991; revised 14 October 1992

Abstract

A new enumeration algorithm is proposed for the propositional satisfiability problem. Such algorithm is based on a hypergraph formulation of the problem. Two different implementations of the algorithm are presented together with the results of an experimentation intended to compare their performance with the performance of other known methods. The computational results obtained are quite promising.

1. Introduction

The satisfiability problem (SAT) is one of the most important combinatorial optimization problems. Since the pioneering work of Cook [5], who has shown it to be the first NP-complete problem, SAT is at the core of the computational complexity theory. Its importance is not only theoretical but also practical, because of its rôle in knowledge-based systems.

Here we present a new approach to its solution based on the use of directed hypergraphs. Directed hypergraphs, a generalization of digraphs, have been introduced in the last years in connection with different types of applications. A systematic presentation of hypergraph concepts and a large number of references are given in [11], where a hypergraph formulation of SAT is presented. From such formulation it follows that SAT can be solved through the solution of a (possibly exponentially large) sequence of easy path problems on hypergraphs. This idea is exploited in this paper in order to devise a new enumeration algorithm, which, from the experimentation performed so far, seems to be quite promising.

The plan of the paper is the following: in Section 2, the main hypergraph concepts are introduced and the hypergraph formulation of SAT is presented; in Section 3, our approach is described; the implementation issues are dealt with in Section 4; in

*Corresponding author.

SSDI 0166-218X(94)00048-4

Section 5 the experimentation performed is described and the results obtained are reported; finally, a few concluding remarks can be found in Section 6.

2. SAT and directed hypergraphs

Let $\mathscr{X}$ be a set of n atomic *propositions*, which can be either *true* or *false*, and denote by T a proposition which is always *true*, and by F a proposition which is always *false*. Let $\mathscr{C}$ be a set of m *clauses*, each of the form

$$x_1 \vee \cdots \vee x_r \leftarrow x_{r+1} \wedge \cdots \wedge x_q, \tag{1}$$

where $x_i \in \mathscr{X}$, for $i = 1, \ldots, q$. The meaning of (1) is that at least one of the propositions $x_1, \ldots, x_r$ must be *true* when all the propositions $x_{r+1}, \ldots, x_q$ are *true*. If this is the case, the clause is *true*; otherwise, if $x_1, \ldots, x_r$ are all *false* and $x_{r+1}, \ldots, x_q$ are all *true*, then the clause is *false*. The disjunction $x_1, \ldots, x_r$ is called the *consequence* of the clause, while the conjunction $x_{r+1}, \ldots, x_q$ is called the *implicant*. We allow for $r = 0$, in which case the consequence is replaced by F, and for $r = q$, in which case the implicant is replaced by T.

A *truth assignment* is a function $v: \mathscr{X} \to \{false, true\}$ that associates a value to each proposition. If a truth assignment which makes all the clauses *true* exists, then $\mathscr{C}$ is said to be *satisfiable*, otherwise it is *unsatisfiable*.

The *satisfiability problem* (SAT) is then defined as follows:

SAT.

Input: a set $\mathscr{X}$ of n propositions, and a set $\mathscr{C}$ of m clauses over $\mathscr{X} \cup \{F, T\}$.
Output: "yes" if $\mathscr{C}$ is satisfiable; "no" otherwise.

Most often, if $\mathscr{C}$ is a satisfiable set of clauses, one wants also a truth assignment which satisfies it. It is easy to see that (1) corresponds to the standard *disjunctive form*

$$x_1 \vee \cdots \vee x_r \vee \neg x_{r+1} \vee \cdots \vee \neg x_q. \tag{2}$$

Hence, each instance $\sigma = (\mathscr{X}, \mathscr{C})$ of SAT corresponds to a propositional formula, S, in *conjunctive normal form*, given by the conjunction of m disjunctions of type (2), one for each clause in $\mathscr{C}$.

2.1. Directed hypergraphs. Cuts and cutsets

We define a directed hypergraph (or simply *hypergraph*) as a pair $\mathscr{H} = (\mathscr{N}, \mathscr{A})$ where $\mathscr{N}$ is the set of nodes and $\mathscr{A}$ is the set of *hyperarcs*. A hyperarc a is a pair (Tail(a), Head(a)), where Tail(a) and Head(a) are two nonempty disjoint subsets of $\mathscr{N}$. These hypergraphs are denominated *2-Graphs* in [11], where a more general and unifying definition of directed hypergraphs is proposed.

We define *size* of hyperarc a the number of nodes it contains, i.e. $|a| = |\text{Tail}(a)| + |\text{Head}(a)|$; the size $\mathscr{S}(\mathscr{H})$ of a hypergraph $\mathscr{H}$ is the sum of the size of its hyperarcs.

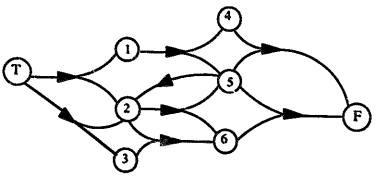

Fig. 1. Hypergraph $\mathscr{H}$.

Let $\mathscr{H} = (\mathcal{N}, \mathcal{A})$ be a hypergraph and s and t be two distinguished nodes. A *cut* $\mathscr{T}_{st} = (\mathcal{N}_s, \mathcal{N}_t)$ is a partition of $\mathcal{N}$ into two subsets $\mathcal{N}_s$ and $\mathcal{N}_t$ such that $s \in \mathcal{N}_s$ and $t \in \mathcal{N}_t$.

Given the cut $\mathscr{T}_{st}$, its *cutset* δ_{st} is the set of all hyperarcs a such that $\mathrm{Tail}(a) \subseteq \mathcal{N}_s$ and $\mathrm{Head}(a) \subseteq \mathcal{N}_t$. The *cardinality* of a cut is the cardinality of its cutset.

Given any instance $\sigma = (\mathscr{X}, \mathscr{C})$ of SAT we can define the hypergraph $\mathscr{H}_\sigma = (\mathcal{N}, \mathcal{A})$ associated with σ; $\mathcal{N}$ and $\mathcal{A}$ are defined as follows:

- $\mathcal{N}$ contains a node for each proposition in $\mathscr{X}$, and two nodes corresponding to T and F;
- $\mathcal{A}$ contains a hyperarc a_C for each clause $C \in \mathscr{C}$; with $\mathrm{Tail}(a_C)$ and $\mathrm{Head}(a_C)$ containing the nodes corresponding to the implicant and the consequence of C, respectively.

As an example, the hypergraph in Fig. 1 corresponds to the following SAT instance:

$$C_1 = x_1 \vee x_2 \leftarrow T,$$

$$C_2 = x_2 \vee x_3 \leftarrow T,$$

$$C_3 = x_4 \vee x_5 \leftarrow x_1,$$

$$C_4 = x_5 \vee x_6 \leftarrow x_2,$$

$$C_5 = x_6 \leftarrow x_2 \wedge x_3,$$

$$C_6 = x_2 \leftarrow x_5,$$

$$C_7 = F \leftarrow x_4 \wedge x_5,$$

$$C_8 = F \leftarrow x_5 \wedge x_6.$$

It is possible to relate cuts in hypergraphs and truth assignments. Consider the hypergraph $\mathscr{H}_\sigma = (\mathcal{N}, \mathcal{A})$ associated with an instance σ of SAT. Given a *TF-cut* $\mathscr{T}_{TF} = (\mathcal{N}_T, \mathcal{N}_F)$ such that $T \in \mathcal{N}_T$ and $F \in \mathcal{N}_F$ it is possible to define the corresponding truth assignment as follows:

$$v(x) = \begin{cases} \textit{true} & \text{if } x \in \mathcal{N}_T, \\ \textit{false} & \text{if } x \in \mathcal{N}_F. \end{cases}$$

Conversely, given any truth assignment v, we can define a cut $\mathscr{T}_{TF}$ as follows:

$$\mathscr{V}_T = \{x: v(x) = true\} \cup \{T\} \text{ and } \mathscr{V}_F = \{x: v(x) = false\} \cup \{F\}.$$

Suppose that there exists a hyperarc a_C in the cutset: it is easy to check that the corresponding clause C is made *false* by v. Hence, a truth assignment satisfying σ corresponds to a cut $\mathscr{T}_{TF}$ with an empty cutset; the following theorem holds (see [11, Theorem 8]).

Theorem 1. *An instance $\sigma \in$ SAT is satisfiable if and only if the associated hypergraph $\mathscr{H}_\sigma$ has a cut $\mathscr{T}_{TF}$ with cardinality* 0.

The satisfiability problems can be restated in terms of cuts in hypergraphs as follows:

SAT_Empty_cut.
Input: a hypergraph $\mathscr{H}_\sigma$ associated with a SAT instance σ.
Output: "yes" if there exists in $\mathscr{H}_\sigma$ a cut $\mathscr{T}_{TF}$ with empty cutset; "no" otherwise.

Actually, many enumerative algorithms for SAT can be interpreted in terms of directed hypergraphs, as algorithms that find a cut with empty cutset, if one exists.

2.2. B-hypergraphs and Horn-SAT

It is well known that SAT is NP-complete; however, many classes of SAT instances are solvable in polynomial time (see, for example, [1, 2, 4, 12]). A widely known example of polynomially solvable SAT instances is the class of Horn formulas (Horn-SAT). The instances in this class contain *Horn clauses*, that is clauses with at most one variable (or the constant F) in the consequence. For this class, linear-time algorithms are known (see [6]).

Hyperarcs corresponding to Horn clauses contain exactly one node in the head; these hyperarcs will be called *backward hyperarcs*, or simply *B-hyperarcs*. A hypergraph in which every hyperarc is a B-hyperarc is called *backward hypergraph*, or *B-hypergraph*; hence the hypergraph $\mathscr{H}_\sigma$ associated with a Horn instance σ is a B-hypergraph. The satisfiability problem for Horn formulas can be easily treated exploiting some of the properties of B-hypergraphs. The reader is referred to [11] for a wider study of B-hypergraphs; in the following we will recall some basic definitions, related to the application to Horn-SAT.

A *path* P_{st} of length q, in the B-hypergraph $\mathscr{H} = (\mathscr{N}, \mathscr{A})$ is a sequence

$$P_{st} = (n_1 = s, a_1, n_2, a_2, \ldots, a_q, n_{q+1} = t),$$

where $n_1, n_2, \ldots, n_{q+1}$ are nodes of $\mathscr{N}$; $a_1, a_2, \ldots, a_q$ are B-hyperarcs of $\mathscr{A}$; and $n_j \in \text{Tail}(a_j)$, $\text{Head}(a_j) = \{n_{j+1}\}$, $j = 1, \ldots, q$.

Nodes s and t are the *origin* and the *destination* of P_{st}, respectively; if $s = t$ then the path P_{st} is a *cycle*. We say that node y is *connected* to a node x in a B-hypergraph $\mathscr{H}$ if a path P_{xy} exists in $\mathscr{H}$.

A *B-hyperpath* Π_{st} in the B-hypergraph $\mathscr{H} = (\mathscr{N}, \mathscr{A})$ is a minimal cycle-free B-hypergraph $\mathscr{H}_\Pi = (\mathscr{N}_\Pi, \mathscr{A}_\Pi)$ such that
(i) $\mathscr{A}_\Pi \subseteq \mathscr{A}$;
(ii) $s, t \in \mathscr{N}_\Pi = \bigcup_{a_j \in \mathscr{A}_\Pi} \mathrm{Tail}(a_j) \cup \mathrm{Head}(a_j) \subseteq \mathscr{N}$;
(iii) $x \in \mathscr{N}_\Pi \Rightarrow x$ is connected to s in $\mathscr{H}_\Pi$.
Node t is *B-connected* to node s in $\mathscr{H}$ if a B-hyperpath Π_{st} exists in $\mathscr{H}$. The following proposition trivially holds.

Proposition 1. *Given a B-hyperpath Π_{st} and a B-hyperarc $a_j \in \mathscr{A}_\Pi$, one has that each node $x \in \mathrm{Tail}(a_j)$ is B-connected to s in Π_{st}.*

A *B-hyperpath* Π from the set of nodes $B \subseteq \mathscr{N}$ to the node t in $\mathscr{H} = (\mathscr{N}, \mathscr{A})$ is a minimal cycle-free B-hypergraph satisfying condition (i) and the following conditions:
(iv) $B \cup \{t\} \subseteq \mathscr{N}_\Pi = \bigcup_{a_j \in \mathscr{A}_\Pi} \mathrm{Tail}(a_j) \cup \mathrm{Head}(a_j) \subseteq \mathscr{N}$;
(v) $x \in \mathscr{N}_\Pi \Rightarrow$ there exists $s \in B$ such that x is connected to s in $\mathscr{H}_\Pi$.
It is possible to relate the concepts of cut and B-connection in B-hypergraphs; in fact, the following theorem holds (Theorem 1 in [11]).

Theorem 2. *In a B-hypergraph $\mathscr{H}$ a cut $\mathscr{T}_{st}$ of cardinality 0 exists if and only if t is not B-connected to s.*

The above theorem can be seen as an extension to B-hypergraphs of a well-known dual relation holding for "standard" directed graphs, where there exists either a path from node s to node t or a cut with empty cutset separating s and t.

A relevant consequence of Theorem 2 is that a Horn instance σ of SAT is satisfiable if and only if in the associated B-hypergraph $\mathscr{H}_\sigma$ F is not B-connected to T. Hence, Horn-SAT can be solved searching a B-hyperpath from T to F in $\mathscr{H}_\sigma$; if such a B-hyperpath is not found, then σ is satisfiable.

The problem of finding a B-hyperpath from T to F can be easily solved by a suitable visit of $\mathscr{H}_\sigma$. The procedure *B-Visit* (see [11]) finds the set of nodes that are B-connected to a given node r in a B-hypergraph; moreover, it returns a set of B-paths containing all the nodes B-connected to r. The time complexity of B-Visit is $O(\mathscr{S}(\mathscr{H}))$; it follows that B-Visit solves an instance of Horn-SAT in linear time.

Actually, *B-Visit*, which bears a strong resemblance with the linear algorithm for Horn-SAT proposed by Dowling and Gallier [6], can be seen as a clever implementation of *positive unit resolution* (i.e., resolution where one of the resolving clauses is restricted to be a *positive unit clause* $x \leftarrow T$; see [17]). When a set $\mathscr{C}$ of Horn clauses is satisfiable, the algorithm above finds the set $\mathscr{V}_T$ of propositions that are implied by $\mathscr{C}$, i.e. propositions x such that $\nu(x) = \mathit{true}$ in each truth assignment ν satisfying $\mathscr{C}$. Once $\mathscr{V}_T$ has been found, an assignment satisfying $\mathscr{C}$ can be obtained simply setting $\nu(x) = \mathit{false}$ for each $x \notin \mathscr{V}_T$. When the formula is not satisfiable, the algorithm finds

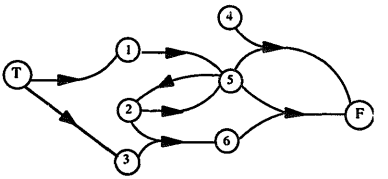

Fig. 2. B-reduction $\mathscr{H}_B^1$ of hypergraph $\mathscr{H}$.

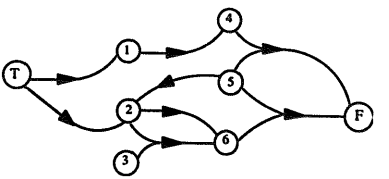

Fig. 3. B-reduction $\mathscr{H}_B^2$ of hypergraph $\mathscr{H}$.

a refutation, i.e. a proof that $\mathscr{C}$ implies $v(F) = true$. In fact, it can be proved that nodes B-connected to T in $\mathscr{H}_\sigma$ correspond to propositions in the set $\mathscr{V}_T$; moreover, a B-hyperpath from T to F corresponds to a refutation of $\mathscr{C}$.

2.3. B-reductions

In this section we will investigate the relations between B-connection in B-hypergraphs and cuts in general hypergraphs, introducing the concept of B-reduction.

Given a hyperarc $a = (\mathrm{Tail}(a), \mathrm{Head}(a))$, a *B-reduction* of a is a B-hyperarc $a_B = (\mathrm{Tail}(a_B), \{v\})$ such that $\mathrm{Tail}(a_B) = \mathrm{Tail}(a)$ and $v \in \mathrm{Head}(a)$. A *B-reduction* of a hypergraph $\mathscr{H}$ is the B-hypergraph $\mathscr{H}_B$ obtained from $\mathscr{H}$ by replacing each hyperarc by one of its B-reductions. As an example, consider the hypergraph $\mathscr{H}$ in Fig. 1; two possible B-reductions of $\mathscr{H}$, $\mathscr{H}_B^1$ and $\mathscr{H}_B^2$, are shown in Figs. 2 and 3, respectively. The hyperarcs corresponding to the non-Horn clauses $\{C_1, C_2, C_3, C_4\}$ have been replaced by one of their B-reduction in order to obtain a B-hypergraph.

Clearly, a hypergraph may have many B-reductions; we shall denote by $\mathscr{B}(\mathscr{H})$ the set of all the B-reductions of $\mathscr{H}$. We say that node t is *super-connected* to node s in hypergraph $\mathscr{H}$ if t is B-connected to s in any B-reduction $\mathscr{H}_B$ of $\mathscr{H}$. Then, t is not super-connected to s if there exists at least one B-reduction in which t is not

B-connected to s. The following theorem (see [11]) relates super-connection and cuts in hypergraphs.

Theorem 3. *In a hypergraph $\mathscr{H} = (\mathcal{N}, \mathcal{A})$ a cut $\mathcal{T}_{st}$ of cardinality 0 exists if and only if t is not super-connected to s.*

The meaning of this theorem in the context of satisfiability is that a SAT instance σ is satisfiable if and only if in the associated hypergraph $\mathscr{H}_\sigma = (\mathcal{N}, \mathcal{A})$, node F is not super-connected to node T, i.e. there exists a B-reduction $\mathscr{H}_B$ of $\mathscr{H}_\sigma$ such that F is not B-connected to T in $\mathscr{H}_B$.

It is easy to see that any given B-reduction $\mathscr{H}_B$ of $\mathscr{H}_\sigma$ can be interpreted as the hypergraph associated with a *Horn subformula* S_B of a formula S; here S_B is obtained by deleting all but one of the propositions from the consequence of every clause in S. Hence, Theorem 3 means that a given formula S is satisfiable if and only if there exists a satisfiable Horn subformula S_B of S.

In fact, it is easy to see that a truth assignment satisfying S_B also satisfies S; moreover, given a truth assignment v satisfying S, we can easily find a Horn subformula S_B satisfied by v, provided that in the B-reduction, $a_B = (\mathrm{Tail}(a), \{n\})$, of a hyperarc a we choose a node $n \in \mathrm{Head}(a)$ such that $v(n) = true$, if one exists (if such a node does not exist, any choice of n will fit).

Theorem 3 suggests a new algorithmic approach for SAT. In fact, a satisfiability problem can be solved by enumerating B-reductions of the associated hypergraph, until a satisfiable one (i.e. one in which F is not B-connected to T) is found, if any. Note that once a B-hyperpath Π is found in a B-reduction that is not satisfiable, then the generation of B-reductions containing Π should be avoided; this fact can provide a guidance for the enumeration.

It is worth noting that in this approach the satisfiability problem reduces to a sequence of easily solvable Horn-SAT problems. A previous attempt in this direction was made by Gallo and Urbani [13]: in their algorithm a *Horn relaxation* of the problem is solved in each node of an enumeration tree. However, the proposed approach is quite different, since here the Horn instances that are solved are not relaxations.

It must be observed that the enumeration of B-reductions could be very inefficient in practice. In fact, the number $|\mathscr{B}(\mathscr{H})|$ of possible B-reductions grows exponentially in the size $\mathscr{S}(\mathscr{H})$ of the hypergraph $\mathscr{H}_\sigma$; moreover, the size $\mathscr{S}(\mathscr{H})$ itself can be exponentially large in the number of variables n. Hence, the number $O(|\mathscr{B}(\mathscr{H})|)$ of B-reductions that must be (implicitly) enumerated could be exponentially larger than the number 2^n of possible truth assignments.

3. A new implicit enumeration algorithm based on B-reductions

Here we present a new implicit enumeration algorithm for SAT, which is based on the use of B-reductions. This algorithm can be considered an implementation of the idea of enumerating the B-reductions discussed in the previous section.

Like all implicit enumeration methods, the proposed algorithm generates a search tree where the root corresponds to the original problem to be solved, and each node corresponds to a subproblem generated from the problem corresponding to the parent node by fixing the value of one or more variables. At each step, the leaf nodes correspond to either subproblems whose solution has been already found (i.e. which have been proved either to be satisfiable or to be unsatisfiable) or to subproblems not yet solved and whose descendants are still to be generated.

The algorithm stops when either a subproblem is recognized to be satisfiable or all the leaf nodes of the current search tree correspond to subproblems which have been recognized to be unsatisfiable. In the former case the answer is "yes" while in the latter the answer is "no'.

The nodes corresponding to subproblems that have not yet been solved and whose descendants are still to be generated are called *active nodes* and are maintained in a priority queue Q; when the algorithm starts, Q contains only the original problem. At each iteration a node is removed from Q and the corresponding subproblem P is examined. In the scanning of Q we use a *depth-first* strategy, i.e. the last node inserted is the one to be removed first.

Each subproblem P has an associated hypergraph $\mathscr{H}_P$ which is obtained from the hypergraph $\mathscr{H}_\sigma$ associated with the original instance σ as follows. Let $\mathcal{N}^T$, $\mathcal{N}^F$, and $\mathcal{N}^*$ be the sets of nodes corresponding to propositions whose value is fixed to *true*, propositions whose value is fixed to *false*, and propositions whose value has not yet been fixed, respectively. We delete from $\mathscr{H}_\sigma$ all the hyperarcs a corresponding to already satisfied clauses (i.e. $\mathrm{Head}(a) \cap \mathcal{N}^T \neq \emptyset$ or $\mathrm{Tail}(a) \cap \mathcal{N}^F \neq \emptyset$), and all the nodes in $\mathcal{N}^T$ ($\mathcal{N}^F$) from the tails (heads) of the remaining hyperarcs; when a tail (head) becomes empty it is replaced by T (F).

A subproblem P is examined in two steps which will be described in detail in Section 4. First we apply a unit resolution algorithm: this step can be considered as a kind of *relaxation operation* and is similar to the basic step of the Horn relaxation algorithm proposed by Gallo and Urbani, where unit resolution is implemented as a forward visit of the hypergraph from the node T; here, the main difference is that the visit is performed also from F, in a backward fashion. Note that the unit resolution is also used as a tool to fix the value of some variables; hence, in this step the hypergraph $\mathscr{H}_P$ associated with R can be further simplified by deleting nodes and hyperarcs as described above. The second step can be interpreted as a *restriction operation* and consists in the generation of a B-reduction $\mathscr{H}_B$ of $\mathscr{H}_P$, and in the search for a B-hyperpath from T to F in $\mathscr{H}_B$. If no such B-hyperpath exists then a TF-cut with empty cutset is found in $\mathscr{H}_P$, and P is proved to be satisfiable. Otherwise, let $\Pi = (\mathcal{N}_\Pi, \mathcal{A}_\Pi)$ a B-hyperpath from T to F in $\mathscr{H}_P$, and let $\{n_1 \dots n_k\}$ be the set of nodes in $\mathcal{N}_\Pi$ different from T and F. It is easy to see that the variables $n_1 \dots n_k$ cannot be all *true* in any truth assignment satisfying P. The set of variables $\{n_1 \dots n_k\}$ is called the *branching set*, and the algorithm generates k new subproblems $P_1, P_2, \dots, P_k$ by fixing the variables according to the following rule:

P_1: $n_1 = \mathit{false}$;
P_2: $n_1 = \mathit{true}$; $n_2 = \mathit{false}$;
P_3: $n_1 = n_2 = \mathit{true}$; $n_3 = \mathit{false}$;
$\vdots$
P_k: $n_1 = \cdots = n_{k-1} = \mathit{true}$; $n_k = \mathit{false}$;

Subproblems $P_1, \ldots, P_k$ replace P in Q. The multiple branching rule used is the one which is widely used in TSP to eliminate subtours (see, for example, [3]). Such rule has also been used in enumerative algorithms for SAT by Monien and Speckenmeyer [19] and by Gallo and Urbani [13].

We can state more formally the resulting algorithm as follows.

Algorithm B-Enum

Step 0: Let P_0 be the problem corresponding to the input instance σ; set $Q = \{P_0\}$;
Step 1: If Q is empty, return "no"; otherwise remove a subproblem P from Q;
Step 2: Apply unit resolution to P; if a contradiction is found, go to Step 1;
if P is solved, return "yes";
Step 3: Select a B-reduction $\mathscr{H}_B$ of the hypergraph $\mathscr{H}_P$ corresponding to P;
if node F is not connected to node T in $\mathscr{H}_B$ then return "yes";
otherwise let $\Pi \subseteq \mathscr{H}_B$ be a B-hyperpath from T to F;
Step 4: Let $\{n_1 \ldots n_k\}$ be the nodes in Π different from T and F;
Add to Q the subproblems $P_1, P_2, \ldots, P_k$ generated from P as follows:
in problem P_i: n_i is set to *false*, and n_j, $1 \leqslant j < i$, are set to *true*.
Goto Step 1.

Note that it is not necessary to use the whole set $\{n_1 \ldots n_k\}$ in the branching step. In fact any set B' such that there exists in $\mathscr{H}$ a subhyperpath of Π from B' to F can be used as the branching set. A B-hyperpath $\pi = (\mathscr{N}_\pi, \mathscr{A}_\pi)$ is a subhyperpath of Π if $\mathscr{A}_\pi \subseteq \mathscr{A}_\Pi$ and $\mathscr{N}_\pi \subseteq \mathscr{N}_\Pi$. As an example, consider the B-reduction $\mathscr{H}_B^1$ in Fig. 2; a B-hyperpath Π from T to F contained in $\mathscr{H}_B^1$ is shown in Fig. 4 (nodes and hyperarcs in Π are shown in thick lines). The set of nodes $B = \{n_3, n_5\}$ gives a valid branching set. In fact, if the corresponding variables are set to *true* in the subproblem corresponding to the hypergraph $\mathscr{H}$ of Fig. 1, a contradiction arises.

A particular branching set is the set of propositions that correspond to the tail of the hyperarc incident with node F in the B-hyperpath Π. Such set is the one which has been used in the implementation of our algorithm.

4. Implementation

In this section the implementation of Step 2 (unit resolution) and Step 3 of the algorithm are described in detail. The resulting procedures use data structures representing directed hypergraphs, that are an extension of those for directed graphs. A hypergraph is represented as a set of lists. For each node n there are two lists, one pointing to the hyperarcs in its *forward star* $\mathrm{FS}(n) = \{a \in \mathscr{A}: n \in \mathrm{Tail}(a)\}$ and one

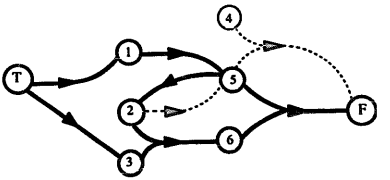

Fig. 4. B-hyperpath Π from T to F in $\mathscr{H}_B^1$.

pointing to the hyperarcs in its *backward star* $\mathrm{BS}(n) = \{a \in \mathscr{A}: n \in \mathrm{Head}(a)\}$. Similarly, for each hyperarc a there are two lists, one pointing to the nodes in $\mathrm{Tail}(a)$ and one pointing to the nodes in $\mathrm{Head}(a)$.

4.1. Unit resolution

As mentioned above, the procedure B-visit can be seen as an efficient implementation of positive unit resolution on Horn instances; it is easy to see that a slight modification of B-visit can perform positive unit resolution on general hypergraphs. Moreover, the resulting procedure can be extended in order to implement *complete* unit resolution: to do this, a *backward* visit of the hypergraph starting from the node F is executed together with the usual forward visit from T.

In practice, a variable n is set to *true* or *false* when an hyperarc $(\{T\}, \{n\})$ or, respectively, $(\{n\}, \{F\})$ is found; the corresponding nodes are deleted from the hypergraph during the visit, together with hyperarcs corresponding to satisfied clauses. Note that deletions are performed *logically*: nodes that are assigned a value are implicitly deleted, while a hyperarc a is deleted by setting a flag $D[a]$ to *true*. For each hyperarc a the counters $T[a]$ and $H[a]$ are used to keep track of deleted nodes in $\mathrm{Tail}(a)$ and $\mathrm{Head}(a)$, in the same way as it is done in B-visit.

The current truth assignment is kept in a vector V; for each node n we have $V[n] \in \{true, false, undefined\}$. Two sets Q^T and Q^F are used to store the nodes to be fixed to *true* and *false*, respectively. The procedure returns the value "inconsistent" if a contradiction is detected. A formal description of the procedure is given below.

It is easy to see that procedure Unit_Resolution has the same complexity of B-visit, i.e. $O(\mathscr{S}(\mathscr{H}))$. So, Step 2 is implemented in linear time, thus extending to general instances the linear-time implementation of the complete unit resolution algorithm proposed by Minoux for Horn formulas [18].

It is possible to prove that a slight modification of procedure Unit_Resolution has an $O(\mathscr{S}(\mathscr{H}))$ *branch complexity*. By branch complexity we mean the worst-case bound for the time spent by the procedure in the examination of all the subproblems belonging to a single *branch*, that is to a path from the root to a leaf in the enumeration tree. A linear branch complexity can be obtained restarting the visit of the hypergraph from the situation left after solving the predecessor of the current subproblem.

```
Procedure Unit_Resolution(𝒦: hypergraph)
 begin
   for each n ∈ 𝒩 do V[n] := undefined;
   for each a ∈ 𝒜 do D[a] := False; T[a] := |Tail(a)|; H[a] := |Head(a)| end_for;
   Q^T := {T}; Q^F := {F};
   repeat
     while Q^T ≠ ∅ do
       select and remove n ∈ Q^T;
       V[n] := True;
       for each a ∈ BS(n) do D[a] := True;
       for each a ∈ FS(n) such that D[a] = False do
         T[a] := T[a] − 1;
         if T[a] = 0 and H[a] = 0 then return (inconsistent);
         if T[a] = 0 and H[a] = 1
         then find n ∈ Head(a) such that V[n] = undefined; Q^T := Q^T ∪ {n} end_if;
         if T[a] = 1 and H[a] = 0
         then find n ∈ Tail(a) such that V[n] = undefined; Q^F := Q^F ∪ {n} end_if;
       end_for
     end_while;
     while Q^F ≠ ∅ do
       select and remove n ∈ Q^F;
       V[n] := False;
       for each a ∈ FS(n) do D[a] := True;
       for each a ∈ BS(n) such that D[a] = False do
         H[a] := H[a] − 1;
         if T[a] = 0 and H[a] = 0 then return (inconsistent);
         if T[a] = 0 and H[a] = 1
         then find n ∈ Head(a) such that V[n] = undefined; Q^T := Q^T ∪ {n} end_if;
         if T[a] = 1 and H[a] = 0
         then find n ∈ Tail(a) such that V[n] = undefined; Q^F := Q^F ∪ {n} end_if;
       end_for
     end_while;
   until Q^F = ∅ and Q^T = ∅;
 end-procedure.
```

4.2. Selection of a B-reduction

The crucial part of the algorithm is the generation of the B-reduction and the search of a B-hyperpath in Step 3. A naïve approach could be to *B-reduce* each hyperarc a, choosing any node belonging to Head(a), and then to apply procedure B-visit. However, it is not necessary to B-reduce all the hyperarcs: we are interested only in the B-hyperpaths contained in the B-reduction, and hence only hyperarcs which can belong to a B-hyperpath need to be B-reduced.

In practice, the generation of a B-reduction can be carried over incrementally while testing the existence of a B-hyperpath: a hyperarc a is B-reduced only if it is *connected*, i.e. if all the nodes in Tail(a) are B-connected to T in the partial B-reduction currently generated. At the beginning of the procedure, the connected hyperarcs are those whose tail contains the unique node T. It is easy to see that a node is B-connected to T if and only if it is chosen as the representative of the head of one of the connected hyperarcs. The incremental building of the B-reduction is performed until either a B-hyperpath from T to F is found (i.e., a B-hyperarc a such that Head(a) = $\{F\}$ becomes connected) or no more connected hyperarcs exist. In the latter case, a feasible B-reduction for the hypergraph can be obtained by selecting any B-reduction for the hyperarcs that are not connected. The resulting procedure can be seen as a variant of procedure B-visit, in which only nodes that are selected are visited. Actually, the procedure finds a set of B-paths contained in the B-reduction; the list L contains the last hyperarc of each one of the detected paths. Note that each B-hyperarc of the type a = (Tail(a), $\{F\}$) belongs at most to one B-hyperpath, that can easily be identified using a *predecessor function* Pred[n]; for each node Pred[n] gives the hyperarc incident with n in a B-hyperpath from T to n. For each hyperarc a, BHead[a] gives the node chosen to B-reduce a; this node is returned by procedure Choose_head(a). The set Q contains the hyperarcs that are connected to T and must hence be B-reduced.

We give next a formal description of the procedure.

```
Procedure B_Reduction (H: hypergraph)
begin
  for each n ∈ N do Pred[n] := null;
  for each a ∈ A do BHead[a] := null; T[a] := |Tail(a)|; endfor;
  L := ∅;
  Q := {a: a ∈ FS(T)};
  while Q ≠ ∅ do
    select and remove a ∈ Q;
    n := Choose_head(a);
    BHead[a] := n;
    if Pred[n] = null then
    begin
      Pred[n] := a;
      for each e ∈ FS(n) such that BHead[e] = null
        T[e] := T[e] − 1;
        if T[e] = 0
        then if |Head(e)| = 0 then L := L∪{e}
                              else Q := Q∪{e};
      endfor
    end
  end_while
end-procedure
```

It is easy to verify that the procedure correctly finds a B-hyperpath in the partial B-reduction generated, if one exists. To evaluate the complexity, notice that the inner "for_each" loops are entered at most $O(\mathcal{S}(\mathcal{H}))$ times, and that each arc is inserted and selected from Q only once.

Observe that a suitable policy for generating feasible B-reductions consists in keeping small the set of nodes B-connected to T. Hence, when B-reducing a hyperarc a, an already chosen node should be preferred, among the ones belonging to Head(a); although not described here, procedure Choose_head exploits this idea. Since B-hyperarcs do not need to be B-reduced, the connected B-hyperarcs are the ones to be considered first; that helps in maintaining small the cardinality of the set of nodes B-connected to T. Following the same line of thought, we can conclude that connected hyperarcs whose heads contain fewer nodes should be considered first. Thus, a hyperarc with the head of minimum size is to be selected from Q at each step. Actually, we implement only partially this idea, since hyperarcs with more than two nodes in the head are selected with the same priority. This allows to perform the selection operation in time $O(1)$.

In practice, the procedure could be stopped when the first B-hyperpath from T to F has been found; the actual implementation behaves as follows: when a B-hyperpath is found, no more hyperarcs are inserted in Q, but the procedure continues until Q becomes empty. The branching set is obtained selecting from L a hyperarc a such that the number of nodes in Tail(a) is minimum.

Two versions of the algorithm have been implemented, which differ in the order in which the nodes are examined in the Choose_head routine and in the branching operation. In the first version, BR1, the ordering of the nodes is arbitrary, while in the second, WBR, the nodes are assigned real weights and the nodes with higher values of the weights are examined first. The weights used are the ones proposed by Jeroslow and Wang [16], whose use has lead to an improved version of Davis and Putnam's algorithm. Such weights are defined for each node n by the formulas:

$$W^T(n) = \sum_{a \in BS(n)} 2^{-|a|},$$

$$W^F(n) = \sum_{a \in FS(n)} 2^{-|a|},$$

and can be considered as a sort of measure of the impact on the problem of fixing the variable n to *true* and to *false*, respectively. In fact, $W^T(n)$ ($W^F(n)$) increases with the number of clauses in whose consequence (implicant) the variable n appears, and decreases with the cardinality of the clauses. Actually, the larger is the number of clauses in which n appears in the consequence (implicant), the higher is the probability that variable n must be set to *true* (*false*) in a satisfying truth assignment, and the larger is the cardinality of a given clause, the less crucial is the rôle of n in making that clause satisfiable.

In WBR, the node chosen as the representative of the head in the B-reduction of a given hyperarc is the node with the minimum $W^T(n)$ value among the ones in the

head, while in the branching operation the branching set is processed in decreasing order of $W^F(n)$, i.e. the node n with higher $W^F(n)$ is fixed to *false* in the first subproblem generated P_1.

5. Computational results

The different versions of the algorithm have been tested on several classes of problems. In order to assess their performance, we have compared them to other existing algorithms, namely to two versions of the Davis and Putnam algorithm, and to the algorithm Horn2 proposed by Gallo and Urbani [13]. The reason of this choice is that such algorithms are considered to be quite efficient (see also Harche et al. [15]) and can be efficiently implemented making use of hypergraph data structure.

Davis and Putnam's algorithm, in the version on which our implementation is based, is an enumerative algorithm in which, when a subproblem P is examined, first unit resolution is applied in order to fix variables and then, if some clauses remain to be satisfied, P is split into two smaller subproblems P^T and P^F by selecting a proposition u and assigning it values *true* and *false*, respectively. The so called *pure literal rule* which is included in the original version of the algorithm (see [17]) has been dropped in our implementation, since it increases the computations without giving any real benefit. The crucial part of the method is the branching criterion, i.e. the selection of n and the order in which P^T and P^F are solved. In the simplest version, DPL/R, the selection of the branching proposition is made randomly, and the subproblem P^T is always solved first. In a second version, DPL/JW, the weights proposed in [16] are used, i.e. the node giving the maximum weight, either $W^T(n)$ or $W^F(n)$, is selected and, respectively, problem P^T or P^F is solved first.

As for Horn2, our implementation differs from the one described in [13] in the fact that the Horn relaxation is solved by the procedure Unit_Resolution described in Section 4.

In order to obtain reasonably hard SAT instances it is necessary to select carefully the distribution model and the size of randomly generated problems. This is not a trivial task, since it is common experience that apparently reasonable distribution models might give quite poor results, that is instances that are very easy to solve (see, for example, [8, 9]). Usually, for a given distribution model, and a fixed number n of variables, the expected difficulty of the instances shows a typical dependence on the number of clauses m: in fact, it is usually possible to find values m_1 and $m_2 > m_1$ such that for $m < m_1$ almost all the instances are satisfiable, while for $m > m_2$ almost all the instances are unsatisfiable. The values m_1 and m_2 are usually quite close to each other, and they define the region in which the most difficult instances lay.

As an example, consider the distribution model used by Fedjki and Hooker [7] where the clauses are generated using the fixed probability model (see [9]), but eventually rejecting unit clauses in order to obtain harder problems. For 100 variables and an average number of literals per clause equal to 5, the most difficult problems are

the ones with m belonging to the interval [700..1000]; out of this range the problems become almost trivial, either because it is easy to find a satisfying truth assignment ($m < 700$), or because inconsistencies arise after fixing a few variables ($m > 1000$).

In addition to the randomly generated problems, we solved a set of *pigeon hole* problems (see [14]), and a set of unsatisfiable 3-SAT instances belonging to the class introduced by Urquhart [20]. The well-known pigeon hole problems consist in assigning $n + 1$ "pigeons" to n "holes", so that no two pigeons share the same hole. These problems are clearly unsatisfiable, and require exponential time to be solved by resolution, as proved by Haken [14]. Urquhart's problems consist in assigning a 0–1 value to the edges of a graph, so that the sum modulo 2 of the values assigned to the arcs incident in each node is equal to the parity value assigned to the node. Urquhart [20] proved that some unsatisfiable problems in this class require exponential time for resolution.

The instances in these classes show a very particular structure (e.g., a pigeon-hole problem contains only positive clauses and negative 2-clauses) and are usually hard to solve also for enumerative methods, becoming intractable for relatively small input sizes.

The results of our computational experiences are listed below. For each distribution and problem size (except for two cases in Table 1) ten instances were solved. We report the average number of nodes in the enumeration tree and the CPU time used for solving the problem (excluding I/O and system time); times are seconds on a HP 9000/720. The algorithms were implemented in standard C language. It is worth noting that the algorithms share the same code for the common parts, and substantially differ only in the implementation of the branching step.

Table 1 contains the results for the problems used in [7]; BR1 and WBR perform better than the other algorithms, and WBR explores fewer nodes than BR1. For these problems, algorithm DPL/JW performs better than Horn2, except in one case (problems with 700 clauses); Horn2 performs better than DPL/R.

In Table 2 we report the results for 3-SAT problems, that substantially confirm the ones of the previous class. Note however that in one case BR1 enumerates more nodes than DPL/JW. For these instances WBR always dominates BR1; DPL/JW always outperforms Horn2, and the latter outperforms DPL/R.

Problems in Table 3 are generated with a discrete uniform model: the number d of literals per clause ranges between 3 and 7 with equal probability. This distribution was chosen in order to obtain the same average $d = 5$ of the problems in Table 1; however the resulting instances are much more difficult. We previously found that a uniform distribution in [2..8] results in very easy problems, probably because many clauses of length 2 are generated.

For these instances BR1 enumerates more nodes than DPL/JW (except for two cases); however, the former takes less CPU time than the latter for instances with more than 1200 clauses, that are the hardest to solve for DPL/JW. Both DPL/JW and BR1 are outperformed by WBR. Also in this case, DPL/JW always outperforms Horn2, and Horn2 outperforms DPL/R.

Table 1
Fixed probability model ($n = 100$)

% of inst.	m	% sat. inst.	Average CPU times (s)				
			DPL/JW	DPL/R	Horn2	BR1	WBR
10	700	100	0.32	1.14	0.17	0.15	0.08
20	800	55	0.49	3.98	1.67	0.25	0.17
20	850	20	0.91	5.30	1.83	0.34	0.34
10	900	10	0.64	4.12	1.37	0.24	0.19
10	1000	0	0.22	0.47	0.32	0.10	0.08
		Total	2.58	15.01	5.36	1.08	0.86
			Average # of nodes enumerated in the search tree				
10	700	100	131	1104	159	70	30
20	800	55	135	2981	1145	89	50
20	850	20	238	3592	1085	117	96
10	900	10	147	2678	680	86	53
10	1000	0	34	214	117	21	13
		Total	685	10569	3186	383	242

Table 2
3-SAT problems ($n = 100$)

m	% sat. instances	Average CPU times (s) – 10 instances				
		DPL/JW	DPL/R	Horn2	BR1	WBR
350	100	0.06	0.34	0.23	0.05	0.04
400	80	1.36	26.65	5.10	0.62	0.63
420	50	2.12	18.40	5.92	1.30	0.79
450	10	1.85	17.37	4.44	1.56	1.00
500	0	1.54	8.98	1.96	1.01	0.62
	Total	6.93	71.74	17.65	4.54	3.08
		Average # of nodes enumerated in the search tree – 10 instances				
350	100	37	517	381	42	21
400	80	859	34455	6303	489	405
420	50	1264	22529	6672	976	496
450	10	991	19578	4603	1073	581
500	0	730	9279	1833	647	329
	Total	3881	86358	19792	3227	1832

From the results of Table 3, one may conjecture that the search strategy based on B-reductions gives worse results, when the number of literals per clause varies in a short interval, with a relatively high average value. This hypothesis is confirmed by the results of 5-SAT problems, listed in Table 4. Note that, in order to solve the

Table 3
Discrete uniform distribution in [3..7] ($n = 100$)

n	% of sat. instance	Average CPU time (s) – 10 instances				
		DPL/JW	DPL/R	Horn2	BR1	WBR
1000	100	0.36	19.99	1.79	0.51	0.17
1100	100	8.86	267.08	24.95	20.29	2.90
1200	80	60.96	2607.01	213.86	124.47	47.33
1250	40	177.85	2159.21	273.80	113.84	57.14
1300	20	167.88	2352.30	233.34	130.72	59.24
1400	10	128.63	1770.25	242.49	94.90	53.62
1500	0	90.59	773.69	117.98	47.75	28.68
	Total	635.13	9949.53	1108.21	532.48	249.08
		Average # of nodes enumerated in the search tree – 10 instances				
1000	100	80	12894	1069	176	28
1100	100	1937	155053	12504	6098	652
1200	80	10995	1264782	88602	32924	8497
1250	40	29851	1182999	104485	27709	9547
1300	20	24751	952728	81201	29057	9067
1400	10	15179	597163	79732	17450	7116
1500	0	8941	220736	28326	7680	3494
	Total	91734	4386355	395919	121094	38401

Table 4
5-SAT problems ($n = 50$)

m	% of sat. instance	Average CPU time (s) – instances				
		DPL/JW	DPL/R	Horn2	BR1	WBR
900	100	1.03	38.60	14.93	13.73	4.96
1000	80	99.14	473.39	130.86	129.40	98.59
1100	10	306.71	951.24	284.86	321.55	195.73
1200	10	261.90	816.57	213.99	270.92	176.58
1300	0	251.36	692.09	201.29	225.06	156.17
	Total	920.14	2971.89	845.93	960.66	632.03
		Average # of nodes enumerated in the search tree – 10 instances				
900	100	371	30051	8854	5636	1618
1000	80	26880	314709	65336	45407	27308
1100	10	69604	552354	123948	96989	46840
1200	10	51195	416655	80373	69302	35650
1300	0	39889	298044	63864	48723	26714
	Total	187939	1611813	342375	266057	138130

problems in a reasonable amount of time, we had to generate problems with $n = 50$ variables.

Actually, DPL/JW enumerates less nodes than BR1, and, in two cases, also less nodes than WBR. However, WBR is outperformed by DPL/JW only for the easy

Table 5
Pigeon-hole problems

# of pigeons	n	m	CPU times (s)				
			DPL/JW	DPL/R	Horn2	BR1	WBR
6	30	81	0.17	0.10	0.07	0.10	0.10
7	42	133	1.68	0.81	0.40	0.70	0.77
8	56	204	19.17	7.45	3.08	6.05	7.07
9	72	297	243.43	82.88	27.70	58.98	65.00
10	90	415	3489.73	988.57	277.38	647.27	727.78
			# of nodes generated in the search tree				
6	30	81	749	423	206	263	263
7	42	133	6491	3081	1237	1663	1663
8	56	204	65561	23313	8660	11523	11523
9	72	297	756687	225129	69281	92 991	92991
10	90	415	9825029	2259295	623530	859911	859911

Table 6
Urquhart's 3-SAT problems

Graph nodes	n	m	Average CPU time (s) – (5 instances)				
			DPL/JW	DPL/R	Horn2	BR1	WBR
10	15	40	0.03	0.02	0.02	0.02	0.03
20	30	80	1.46	0.76	0.62	0.55	0.68
30	45	120	54.44	25.16	16.76	11.07	14.38
40	60	160	2176.73	795.18	425.66	182.33	227.87
			Average # of nodes generated in the search tree (5 instances)				
10	15	40	127	127	121	119	119
20	30	80	4095	4543	3846	2582	2582
30	45	120	131071	148019	104644	50687	50687
40	60	160	4194303	4510994	2637355	809499	809499

problems with 900 clauses. Note that for this class both BR1 and DPL/JW are outperformed by Horn2, even if the latter enumerates more nodes.

Table 5 contains the results for the pigeon-hole problems. The results for Urquhart's problems are listed in Table 6: we generated instances corresponding to 4 graphs, with 10 up to 40 nodes; for each graph, we generated 5 instances, assigning different parity values to the nodes.

Note that, due to the particular structure of these problems, some algorithms show a "pathological" behaviour: in fact, DPL/JW always gives the worst results, and is outperformed even by DPL/R. This kind of behaviour was observed also in [15], for several particular classes of problems. Furthermore, DPL/JW enumerates $2^{(n/3)+2} - 1$ nodes when applied to a Urquhart's problem with n variables. On the contrary, Horn2

Table 7
Average execution time (sat. and unsat. instances)

Model		% of sat. instances	DPL/JW	DPL/R	Horn2	BR1	WBR	
Fixed								
Prob	$n = 100$	55	0.38	4.24	1.69	0.21	0.15	sat
	$m = 800$		0.64	3.68	1.64	0.30	0.20	unsat
3-SAT	$n = 100$	50	0.80	9.64	2.82	0.37	0.27	sat
	$m = 420$		3.45	27.17	9.02	2.22	1.32	unsat
[3..7] SAT	$n = 100$	40	135.86	2159.21	141.90	47.38	32.65	sat
	$m = 1250$		205.84	3004.21	361.74	158.15	73.47	unsat
5-SAT	$n = 50$	80	33.07	241.06	77.93	59.11	57.15	sat
	$m = 1000$		363.42	1402.70	342.58	410.57	264.31	unsat

is much more stable, and gives the best results for pigeon-hole problems. It is worth noticing that for these problems algorithms BR1 and WBR enumerate the same number of nodes.

In order to give an explanation of the experimental results, first note that the B-reduction method, as implemented in BR1 and WBR, can be seen as an extension of algorithm Horn2. In fact, the two methods share an equivalent branching rule, which is, respectively, applied to hyperarcs incident with node F and with node T. In addition, algorithms BR1 and WBR incorporate a heuristic for finding solutions (Step 3) and a selection rule for the branching step. The underlying idea of the selection rule is to concentrate on those situations in which a potential contradition, i.e. a B-path, has been detected. This contradiction may be ruled out by branching, or may lead to an early detection of the unsatisfiability of the subproblem.

The selection rule based on the search of a B-reduction proves to be almost as effective as the one based on Jeroslow and Wang's weights; in fact, even the simpler implementation of our method, BR1, compares favourably with algorithm DPL/JW. Actually, BR1 is slightly worse than DPL/JW only for 5-SAT problems; for this class the selection rules turn out to be too expensive, and both BR1 and DPL/JW are outperformed by Horn2. As one might expect, if the search of B-reductions takes advantage of weights a much faster algorithm is obtained. Note however that for the problems in Tables 5 and 6 the weights do not provide useful informations; nevertheless, the use of B-reductions is still useful, at least for Urquhart's problems.

Computational results show that the B-reduction method is not always the best for easy satisfiable instances. In Tables 7 and 8 we investigate the relationship between the behaviour of the algorithms and the satisfiability of the problems. For each one of the classes in Tables 1–4 we choose the group of problems which are satisfiable in about half of the cases. In order to allow for meaningful comparison, in Table 7 we report separately the average execution times of the algorithms on the satisfiable instances and on the unsatisfiable ones. In Table 8, for each class, we report the above times

Table 8
Normalized average execution time (sat. and unsat. instances)

Model		% of sat instances	DPL/JW	DPL/R	Horn2	BR1	WBR	
Fixed Prob	$n = 100$	55	7.60	10.64	10.12	8.30	8.66	sat
	$m = 800$		12.92	9.22	9.86	12.08	11.84	unsat
3-SAT	$n = 100$	50	3.77	5.24	4.77	2.88	3.36	sat
	$m = 420$		16.23	14.76	15.23	17.12	16.64	unsat
[3..7] SAT	$n = 100$	40	7.64	8.10	5.18	4.16	5.71	sat
	$m = 1250$		11.57	11.27	13.21	13.89	12.86	unsat
5-SAT	$n = 50$	80	3.34	5.09	5.96	4.57	5.80	sat
	$m = 1000$		36.66	29.63	26.18	31.73	26.81	unsat

normalized with respect to the total execution time for the problems in that class. The expected normalized value is 10; a large difference between the two values corresponding to the same group reveals a strong dependence on the satisfiability of the problem.

The results in Table 7 substantially confirm the previous observations, and do not show any pathological dependence. From Table 8 it turns out that WBR is more stable than BR1, i.e. it shows a weaker dependence on the satisfiability of the problems: this can be explained by the more effective selection rule. Note also that DPL/R and Horn2, which do not incorporate any selecting or guessing heuristic, are usually stable.

6. Conclusions and further work

In this paper the satisfiability problem is stated in terms of directed hypergraphs. The resulting formulation is then reduced to the problem of finding a particular B-reduction. An enumerative algorithm that follows this approach is proposed, and two different implementations are tested. It seems apparent from the computational results that the proposed approach is effective on a wide range of problems. It would be interesting to verify its behaviour on practical problems, such as instances arising from large non-Horn propositional databases.

It should be observed that the present versions of the algorithm are based on a relatively straightforward application of results arising from the theory of hypergraphs. Further developments are possible; in particular, a deeper investigation of the structure of the hypergraph model may suggest algorithmic improvements, while the use of reoptimization techniques may lead to faster implementations. An interesting open problem, which we would like to address in the future, is the one of choosing the most effective branching set among the ones existing in a given B-hyperpath.

References

[1] B. Aspvall, Recognizing disguised NR(1) instances of the satisfiability problem, J. Algorithms 1 (1980) 97–103.

[2] B. Aspvall, M.F. Plass and R.E. Tarjan, A linear time algorithm for testing the truth of certain quantified Boolean formulas, Inform. Process. Lett. 8 (1979) 112–123.

[3] E. Balas and P. Toth, Branch and bound methods, in: E.L. Lawler, J.K. Lenstra, A.H. Rinnooy Kan and D.B. Shmoys, eds., The Traveling Salesman Problem (Wiley, New York, 1985).

[4] Chandru and Hooker, Extended Horn sets in propositional logic, J. ACM 38 (1991) 205, 221.

[5] S.A. Cook, The complexity of theorem proving procedures, in: Proceedings of the ACM Symposium on the Theory of Computing (ACM, New York, 1971) 151–158.

[6] W.F. Dowling and J.H. Gallier, Linear-time algorithms for testing the satisfiability of propositional Horn formulae, J. Logic Programming 1 (1984) 267–284.

[7] C. Fedjki and J. N. Hooker, Branch and cut solution of inference problems in propositional logic, Ann. Math. Artificial Intelligence 1 (1990) 123–140.

[8] J. Franco, On the probabilistic performance of algorithms for the satisfiability problem, Inform. Process. Lett. 23 (1986) 103–106.

[9] J. Franco and M. Paull, Probabilistic analysis of the Davis–Putnam procedure for solving the satisfiability problem, Discrete Appl. Math. 5 (1983) 77–87.

[10] M.R. Garey and D.S. Johnson, Computers and Intractabilty (Freeman, San Francisco, CA, 1978).

[11] G. Gallo, G. Longo, S. Nguyen and S. Pallottino, Directed hypergraphs and applications Discrete Appl. Math. 42 (1993) 177–201.

[12] G. Gallo and M.G. Scutellà, Polinomially solvable satisfiability problems, Inform. Process. Lett. 29 (1988) 221–227.

[13] G. Gallo and G.P. Urbani, Algorithms for testing the satisfiability of propositional Horn Clauses, J. Logic Programming 7 (1989) 45–61.

[14] A. Haken, The intractability of resolution, Theoret. Comput. Sci. 39 (1985) 297–308.

[15] F. Harche, J.N. Hooker and G.L. Thompson, A computational study of satisfiability algorithms for propositional logic, Working Paper 1991-27, Graduate School of Industrial Administration, Carnegie Mellon University.

[16] R.G. Jeroslow and J. Wang, Solving propositional satisfiability problems, Ann. Math. Artificial Intelligence 1 (1990) 167–187.

[17] D. Loveland, Automated Theorem-Proving: A Logical Basis (North-Holland, Amsterdam, 1978).

[18] M. Minoux, The unique Horn-satisfiability problem and quadratic Boolean equations, Ann. Math. Artificial Intelligence 6 (1992) 253–266.

[19] B. Monien and E. Speckenmeyer, Solving satisfiability in less than 2^n steps, Discrete Appl. Math. 10 (1985) 287–295.

[20] A. Urquhart, Hard examples for resolution, J. ACM 34 (1987) 209–219.